g-2 trolley system
Final Internship Report

Matteo Bartolini

September 25, 2015

Contents

1 Introduction to g-2 experiment

We know from physics that the magnetic moment of a particle is related to its spin by the following equation:

$$\vec{\mu} = g(\frac{q}{2m})\vec{S} \tag{1}$$

where q and m are the charge and the mass of the particle and g is the gyromagnetic factor which, for a structurless $\frac{1}{2}$ spin particle, is expected to be 2 from Dirac's equation.
The difference g-2 is due to radiative corrections in QED and has been determined with increasing precision in the last decades both theoretically and experimentally. The Standard Model of particle physics makes a very precise prediction to the value g-2, accurated to 400 ppb. The most recent measurement was performed at Brookhaven National Laboratory (experiment E821) and achived a precision of about 540 ppb.The final result is:

$$a_\mu^{E821} = (11659208.0 \pm 6.3) * 10^{-10} \tag{2}$$

QED calculations give:

$$a_\mu^{SM} = (11659182.8 \pm 4.5) * 10^{-10} \tag{3}$$

The difference Δa_μ is 3.3σ. In order to be confident that this is not a statistical fluctuation, a difference of 5σ is required.
The purpose of Fermilab g-2 experiment is to achive a precision of 140 ppb. With this increased precision we can better compare the difference between the theoretical and the experimental value and this should provide an answer to the question whether there are new forces and particles which may exist in nature.
In all the experiments that have been performed so far muons have been used. In the SM effects on the magnetic moment scale with power of m_l^2 which makes muons more suitable for this purpose, since they are 207 times heavier than electrons. For this measure, muons are injected into a storage ring where they follow a circular orbit with a lifetime of about 64 microseconds before decaying into electron and neutrinos.

2 Evaluation of a_μ with SM

According to the Standard Model the anomaly can be described as a sum of the hadronic and the elecroweak terms:

$$a_\mu = a_\mu^{QED} + a_\mu^{EW} + a_\mu^{Hadronic} \tag{4}$$

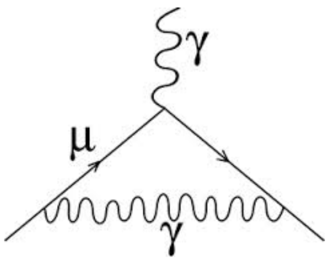

Figure 1: qed loop

The QED and EW terms have been calculated with very high precision using perturbation theory, thanks to the small value of the coupling costant $\frac{\alpha}{4\pi}$. The hightest uncertanty in this theoretical calculation comes from the hadronic term, since it cannot be calculated using perturbation theory at low energy. The QED contribution includes photonic loop and is calculated analitically up to the third order.
The EW term includes contribution from Z and W bosons. It is calculated up to the second order.

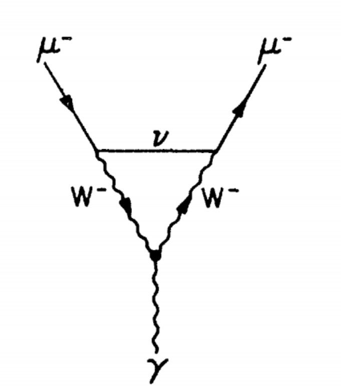

Figure 2: Weak contribution to the muon anomalous magnetic moment. This figure shows a one-loop diagram with virtual W bosons

The hadronic term includes quarks and gluons loops and gives $a_\mu^{Hadronic} = (6945 \pm 62) * 10^{-11}$

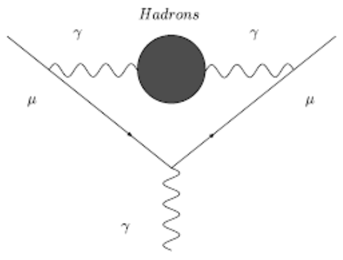

Figure 3: Feynman diagram for an hadronic loop

3 Experimental method

3.1 Producton of muons

Muons are produced by sending a proton beam into a target material. This gives origin to positive and negative pions whose primary decay, with a probability of 0.999877, is purely leptonic and creates muons and neutrinos:

$$\pi^+ \longrightarrow \mu^+ + \nu_\mu \tag{5}$$

$$\pi^- \longrightarrow \mu^- + \bar{\nu_\mu} \tag{6}$$

Since in nature there are almost only left-handed neutrinos and right-handed antineutrinos, due to momentum and spin conservation law, the produced muons will all have a defined polarization (projection of the spin $\vec{S}$ along the momentum $\vec{p}$ direction). These muons are injected into a storage ring where a magnetic field is present.

3.2 Muons in the magnetic field

In the magnetic field muons move along the ring with the cyclotron frequency $\omega_C = \frac{qB}{\gamma mc}$ given by the equation of motion:

$$\frac{d\vec{p}}{dt} = q\vec{v} \times \vec{B} \tag{7}$$

On the other and the interaction of the muon magnetic moment $\vec{\mu}$ with the magnetic field $\vec{B}$ gives origin to the Larmor frequency $\omega_S = \frac{gqB}{2mc} + (1-\gamma)\frac{qB}{\gamma mc}$ The anomalous precession frequency is determined from the difference:

$$\omega_a = \omega_C - \omega_S = \frac{(g-2)}{2}\frac{qB}{m} = a_\mu \frac{qB}{m} \tag{8}$$

In order to provide vertical focusing electrical quadrupoles are also used. In the muon rest frame this is seen as a magnetic field that can affect the Larmor frequency so a correction to the previous expression for ω_a must be made:

$$\vec{\omega_a} = a_\mu \frac{q\vec{B}}{m} + (a_\mu - \frac{1}{\gamma^2 - 1})\frac{\vec{\beta} \times \vec{E}}{c} \tag{9}$$

Looking at this equation we can see that the term $(a_\mu - \frac{1}{\gamma^2-1})$ vanishes at the magic momentum of 3.094 $\frac{Gev}{c}$ which correspond to $\gamma = 29.6$. Thus a_μ can be obtained by a precision measurement of $\vec{B}$ and ω_a As already said, muons in the ring live, on average, 64 microsecons before decaying via an electroweak process into electrons and neutrinos:

$$\mu^+ \longrightarrow e^+ + \bar{\nu_\mu} + \nu_e \tag{10}$$

$$\mu^- \longrightarrow e^- + \bar{\nu_e} + \nu_\mu \tag{11}$$

We know that parity is conserved in electromagnetic and strong interaction, but not in a electroweak process. This means that a correlation between the spin orientation of the muon and the momentum direction of the emitted electron(positron) exist. In other words there is a preferred direction for the emission of the electron for each spin orientation of the muon. In the muon rest frame the differential probability for an electron to be emitted with a normalized energy $y = \frac{E}{E_{max}}$ (where $E_{max} = 52$ Mev is the maximum energy that can be transfered to the electron) at an angle θ with respect to the muon spin is:

$$\frac{dP(y,\theta)}{dyd\Omega} = \frac{1}{2\pi}n(y)[1 \pm \alpha(y)cos\theta] \qquad with \tag{12}$$

$$n(y) = y^2(3-2y) \qquad and \tag{13}$$

$$\alpha(y) = \frac{q}{e}\frac{2y-1}{3-2y} \tag{14}$$

Where + is used for positrons and - for electrons. Positrons and electrons with $y \geq 0.5$ tend to emerge in the direction parallel and antiparallel to the muon spin respectively. In the lab frame the distribution rotates at the angular frequency ω_a

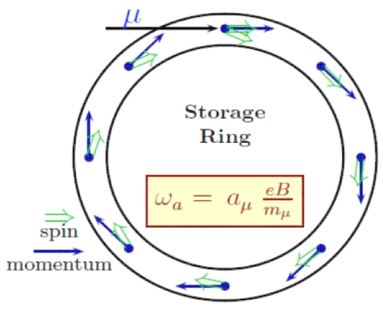

Figure 4: This figure shows the momentum and the spin precession along the ring. The muon spin axis changes 12 degrees with respect to the momentum axis after each rotation

3.3 Measurement of ω_a

We have already seen from eq (8) that, in order to measure a_μ, one needs to know B and ω_a. Now we will talk about how measure ω_a. In order to minimize the uncertanty in the measure of ω_a only positrons with an energy over E_{th} are selected. Consequently the integrated number of positrons above E_{th} is modulated at the frequency ω_a. The expression is shown to be:

$$N(t) = N_0 exp(-\frac{t}{\tau\gamma})[1 - Acos(\omega_a t + \phi] \tag{15}$$

In this experiment positron detection is made with photomultipliers distributed along the ring. These detectors measure the energy of positrons. However the trajectory of these particles must also be found to reconstruct the point where the muon decayed. The muon position information is particularly important in characterizing the megnetic field felt by the muon at the moment of its decay.

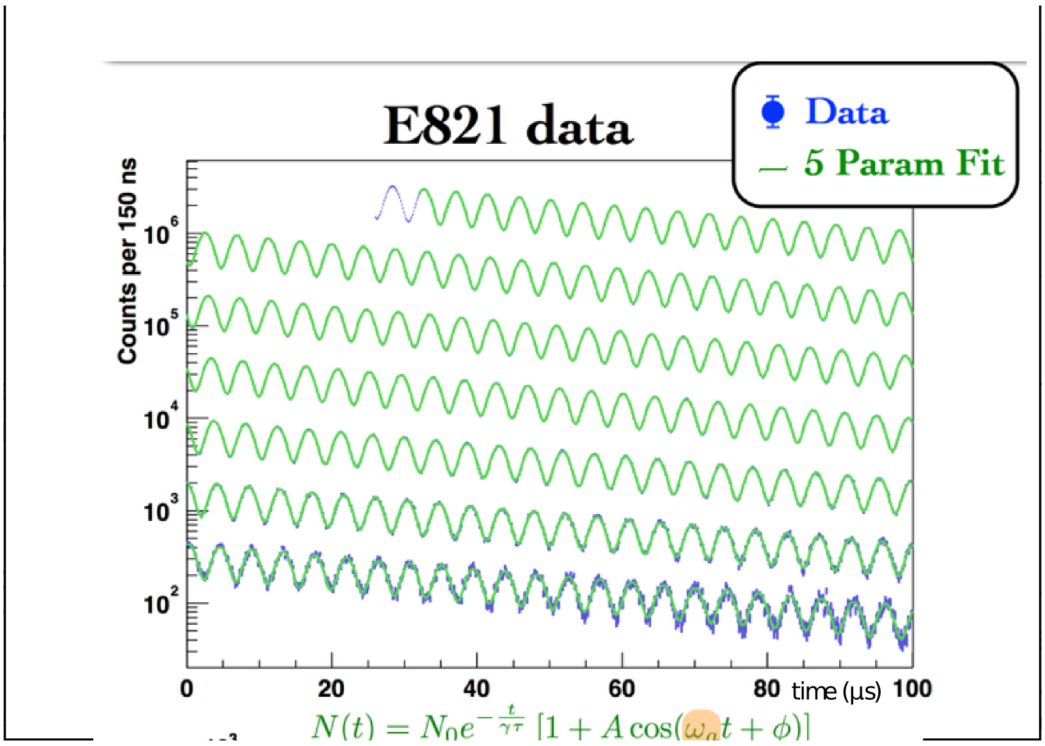

Figure 5: proton frequency

3.4 Measurment of the magnetic field

The magnetic field is measured using NMR. A trolley system made of 17 sferical water probes scans the field at 6000 azimutal locations around the ring to obtain the correspondent proton precession frequency ω_p. It can be shown that a_μ is linked to $\tilde{\omega_p}$,the proton precession frequency averaged over the ring, through the following expression:

$$a_\mu = \frac{R}{\lambda - R} \tag{16}$$

where $R = \frac{\omega_a}{\omega_p}$ and $\lambda = \frac{g_\mu m_p}{g_p m_\mu}$ is the muon to proton magnetic moment ratio, determined from muonium hyperfine level structur measurments. This last value is known to 8 ppb. In the Brookhaven experiment sytematic uncertanties for the ω_p were estimated to be 170 ppb. FNAL goal is 70 ppb. In the next chapters i will go into detail about how the measure of $\vec{B}$ is carried out, since this was part of my work at Fermilab.

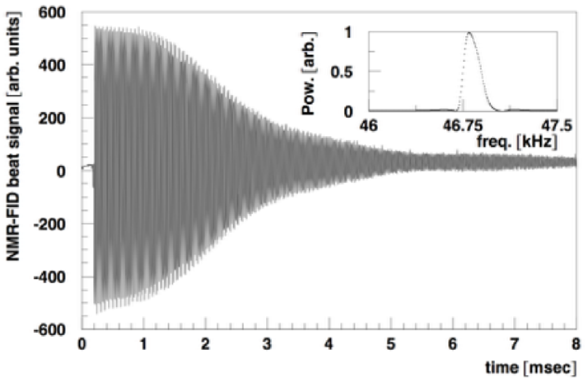

Figure 6: proton frequency

4 Storage Ring

The ring used in this experiment is the one used at Brookhaven National Laboratory. It was moved 3200 miles by land and sea to Fermilab from New York during summer 2013. This C shaped ring has a 7 metres radius and is designed to have a very homogeneous vertical field of about 1.4 Tesla. A very uniform field, generated by the superconducting coils,reduces requirements on the knowledge of the location of the magnetic probes. For the purpose of this experiment the field should be know to 0.1 ppm. The magnet is also designed as a shimmable kit. Passive iron shimming is used to correct imperfections in the initial assembly by a factor of two or three order of magnitude. In the following picture a cross section of the storage ring is shown:

G. W. BENNETT *et al.*

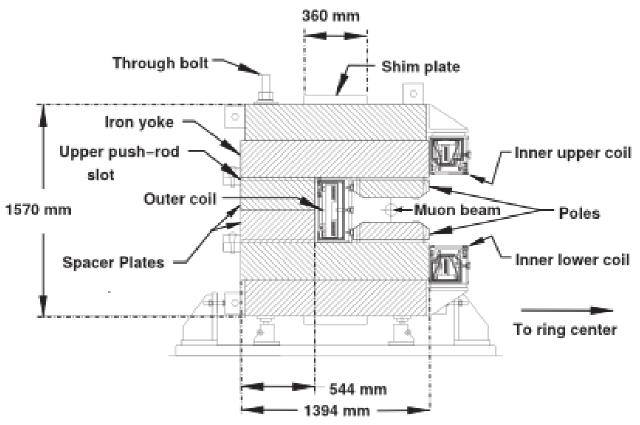

FIG. 7. Cross sectional view of the C magnet.

Figure 7: Cross section of the magnet

4.1 Inflector

The inflector is a superconducting magnet whose vertical fiels cancels the main storage field allowing muons to pass largely undeflected into the ring. The muons coming from the inflector require a pulsed kicker to be put into the phase space acceptance of the ring. The center of the circular orbit for the just injected muon is offset from that of the storage ring and this would lead to the particle being lost after one revolution.

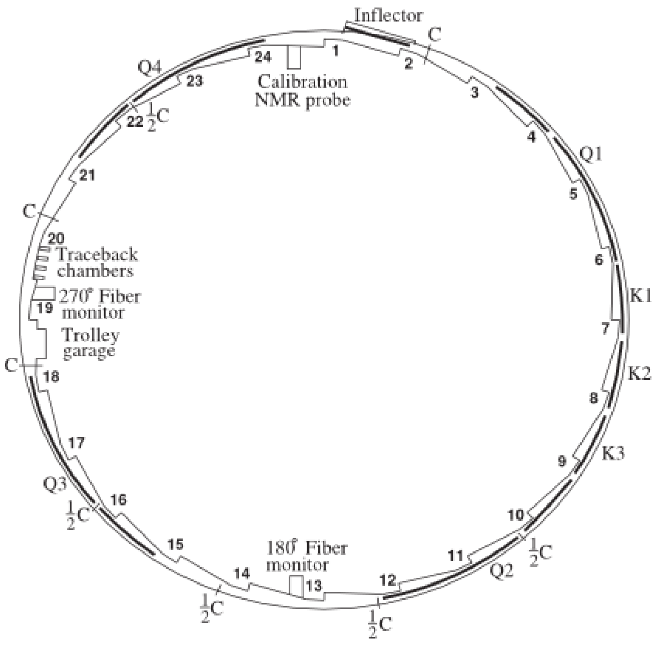

Figure 8: Cross section of the magnet

5 Trolley System

The trolley is a system that is used to measure the magnetic field around the ring, as already said. All the 17 probes that it contains must be calibrated using a very homogeneus magnetic field. In this experiment, for the purpose, a solenoid of a MRI machine is used. The steps taken for calibration are listed below:

- Put absolute sferical water probe in MRI solenoid to measure B_{abs}
- Measure the field using the trolley system to get $B_{trolley}$
- Take the difference between the two to obtain $\Delta B = B_{abs} - B_{trolley}$

During my experience at Fermilab i was involved in the development of an automated system to bring the water probes in and out of the solenoid upon request and of an interface with MIDAS, the data acquisition system used for this experiment. The basic idea is to program a commercial controller called Galil to control motors movement using C/C++ languages. It is important to have a stable and precise system in order to reduce systematc errors. Absolute water probes allow to measure B-field with a precision to 20 ppb, but grandients in the field may affect the accuracy if an error in the probes dispacement is made: $\Delta\vec{B} = \vec{B}_0 + \frac{\partial\vec{B}}{\partial\vec{x}} \cdot \Delta\vec{x}$ Another crucial improvement that must be achieved is the resolution in the azimutal position measurment along the ring. This experiment aims at having a resolution of less than 2mm compared to the 2cm resolution of the previous experiment at Brookhaven.

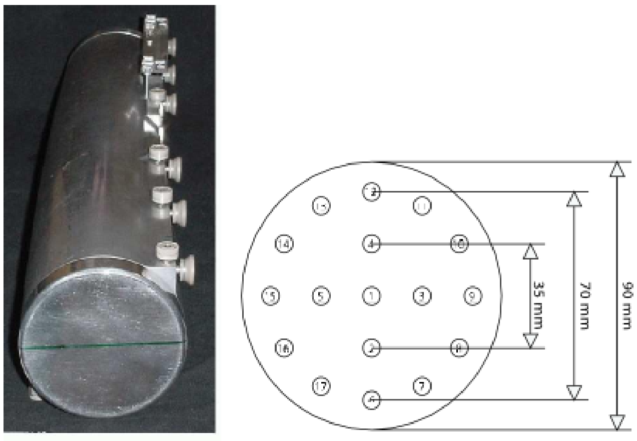

Figure 9: View of the trolley and the probes

5.1 Galil board

This is a commercial controller produced by the company Galil Motion Control. The one used for this experiment has the following characteristics:

- It can control up to six exes at the same time
- Each axis has 26 pins to comunicate with the motors
- It can be programmed using the Galil interface or using c++ libraries downloaded from the company website

The functions defined in these libraries can be used in any C++ code to send command or store the output. In particular, this functions will be called by the software MIDAS at the right time. Here i will report the ones that i used and a quick explanation of what they do:

- Gopen() which allows the user to set up a communication with the controller.
- GInfo() which returns information about Galil
- GCommand() sends commands to galil and stores the answer into the buffer.

- GProgramDownload() allows to download into Galil a sequence of commands
- GProgramDownloadFile() allows to download a text file into Galil.
- GMessage() is used to read outputs from Galil, tipically when using the command MG in a Galil script.
- GCmd() sends command to Galil but does not store any output coming from the controller.

5.2 MIDAS

The g-2 DAQ is being developed using the MIDAS data acquisition software package, which was first created at PSI and now is widely used in many another labs. It will process data from 1296 calorimeters channel, 3 straw tracker station and multiple auxiliary detectors at the expected rate of 18 $\frac{GB}{s}$.
MIDAS provides a convenient web interface for control of the experiment, as shown in fig ,as well as the framework for an event builder and data logger, which will output data in a MIDAS binary format, which can subsequently be processed into a ROOT tree and analyzed. MIDAS also provides an online database (ODB) used both for saving the configuration of the experiment from run-to-run and also for control of the detectors, as settings that are changed in the ODB are hot-coded to update via the frontend processes in real time. A frontend application consists of:

- a fixed experiment-independent system framework handling the data flow control, data transmission and run control operation.
- a user part

The code that i wrote allows to implement a sequence of movement and to monitor many motors parameters such as position, speed, acceleration, torque ecc. Data acquired are stored into ROOT files. This code can be improved to prevent motor from getting damaged if something goes wrong during a run. I have not been able to do it due to the fact that motor were under repair for a long time and i couldn't use them for testing. Here i post the code written by me that allows to talk to Galil from MIDAS web interface.

```
/*********************************************************************\

  Name:          frontend.c
  Created by:   Matteo Bartolini

  Contents:     readout code to talk to Galil motion control

  $Id$

\*******************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "midas.h"
#include "mcstd.h"
#include "experim.h"
#include "gclib.h"
#include "gclibo.h"
#include <iostream>
#include <string>
#include <iomanip>
#include "/home/galil/DAQ/midas/drivers/device/nulldev.h"
#include "/home/galil/DAQ/midas/drivers/bus/null.h"
#include "/home/galil/DAQ/midas/drivers/class/hv.h"
#include "/home/galil/DAQ/midas/drivers/bus/rs232.h"
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/timeb.h>
#include <fstream>
#include <sstream>


#define GALIL_EXAMPLE_OK G_NO_ERROR //return code for correct code exe
#define GALIL_EXAMPLE_ERROR -100
using namespace std;

/* make frontend functions callable from the C framework */
#ifdef __cplusplus
extern "C" {
#endif

  ofstream myfile;
  // i am defining some Galil libraries variables
 INT level1=2;
  float axes[3];
  INT setaxes[3];
  float speed[3];
  float acceleration[3];
  float torque[3];
  INT getaxes[3];
  HNDLE hDB, hkeyclient;
 char  name[32];
  int   size; //size of axes[3]
  INT size1; // size of setaxes[3]
  INT allow;

  int i;
  string s;
  int s1;
 GReturn b = G_NO_ERROR;
        int rc = GALIL_EXAMPLE_OK; //return code
        char buf[1023]; //traffic buffer
  char buf1[1024];
  //char buf2[1024];
        GCon g = 0; //var used to refer to a unique connection. A valic
  //----------------------------------------------------------------


/*-- Globals -------------------------------------------------------*/

/* The frontend name (client name) as seen by other MIDAS clients
*/
char *frontend_name = "Sample Frontend";
/* The frontend file name, don't change it */
char *frontend_file_name = __FILE__;

/* frontend_loop is called periodically if this variable is TRUE
*/
BOOL frontend_call_loop = FALSE;

/* a frontend status page is displayed with this frequency in ms */
INT display_period = 3000;
```

```
/* maximum event size produced by this frontend */
INT max_event_size = 10000;

/* maximum event size for fragmented events (EQ_FRAGMENTED) */
INT max_event_size_frag = 5 * 1024 * 1024;

/* buffer size to hold events */
INT event_buffer_size = 100 * 10000;


/*-- Function declarations -----------------------------------------*/

INT frontend_init();
INT frontend_exit();
INT begin_of_run(INT run_number, char *error);
INT end_of_run(INT run_number, char *error);
INT pause_run(INT run_number, char *error);
INT resume_run(INT run_number, char *error);
INT frontend_loop();

  INT read_galil_event(char *pevent, INT off);
  INT read_trigger_event(char *pevent, INT off);

INT poll_event(INT source, INT count, BOOL test);
INT interrupt_configure(INT cmd, INT source, POINTER_T adr);
  INT db_set_value(HNDLE hDB, HNDLE hKeyRoot, const char *key_name, co
  INT db_find_key(HNDLE hdB, HNDLE hKey, const char *key_name, HNDLE *
  INT cm_get_experiment_database(HNDLE *hDB, HNDLE *hKeyClient);


/* device driver list */
/*DEVICE_DRIVER hv_driver[] = {
  {"Dummy Device", nulldev, 16, null},
  {""}
  };*/


/*-- Equipment list ------------------------------------------------*/
EQUIPMENT equipment[] = {

     {"Galil",                  /* equipment name */
    {3, 0,                      /* event ID, trigger mask */
     "SYSTEM",                  /* event buffer */
     EQ_PERIODIC,               /* equipment type */
     0,                         /* event source */
     "MIDAS",                   /* format */
     TRUE,                      /* enabled */
     RO_RUNNING | RO_TRANSITIONS |    /* read when running and on trans
     RO_ODB,                    /* and update ODB */
     1000,                     /* read every 1 sec */
     0,                         /* stop run after this event limit */
     0,                         /* number of sub events */
     1,                         /* log history */
     "", "", "",},
    read_galil_event,          /* readout routine */
    },



   {""}
};


#ifdef __cplusplus
}
#endif

/********************************************************************\
               Callback routines for system transitions

  These routines are called whenever a system transition like start/
  stop of a run occurs. The routines are called on the following
  occations:

  frontend_init:  When the frontend program is started. This routine
                  should initialize the hardware.

  frontend_exit:  When the frontend program is shut down. Can be used
```

```
                  to releas any locked resources like memory, commu-
                  nications ports etc.

  begin_of_run:   When a new run is started. Clear scalers, open
                  rungates, etc.

  end_of_run:     Called on a request to stop a run. Can send
                  end-of-run event and close run gates.

  pause_run:      When a run is paused. Should disable trigger events.

  resume_run:     When a run is resumed. Should enable trigger events.
\*************************************************************/

/*-- Frontend Init ------------------------------------------*/

INT frontend_init()
{
 myfile.open("/home/galil/experiment/galilmove.dmc");
 myfile <<"#MOVE\nKIA=0.1\nKPA=103\nKDA=2268\nSPA=400\nACA=20000\nTLA=
 myfile.close();





        b=GOpen("/dev/ttyUSB0 -t 1000 -s MG -d", &g);
        //GOpen("192.168.1.42 -s ALL -t 1000 -d",&g);
        //GOpen("00:50:4c:38:19:AA -s ALL -t 1000 -d", &g);
        GInfo(g, buf, sizeof(buf)); //grab connection string
        cout << "buf is" << " "<< buf << "\n";
        if (b==G_NO_ERROR){
        cout << "connection succesfull\n";
        }
        else {cout << "connection failed \n";}

        GProgramDownload(g,"",0); //to erase prevoius programs
         b=GProgramDownloadFile(g,"/home/galil/experiment/galilmove.dm
        GCmd(g, "XQ");

        GTimeout(g,2000);//adjust timeout
        //int i = 0;
        //int s;

        //---------------end code to communicate with Galil---------------


   return SUCCESS;
}

/*-- Frontend Exit -------------------------------------------------*/

INT frontend_exit()
{
   return SUCCESS;
}

/*-- Begin of Run -------------------------------------------------*/

INT begin_of_run(INT run_number, char *error)
{

  return SUCCESS;
}

/*-- End of Run ---------------------------------------------------*/

INT end_of_run(INT run_number, char *error)

{



   return SUCCESS;
}

/*-- Pause Run ----------------------------------------------------*/

INT pause_run(INT run_number, char *error)
{
   return SUCCESS;
}

/*-- Resuem Run --------------------------------------------------*/

INT resume_run(INT run_number, char *error)
{



   return SUCCESS;
}

/*-- Frontend Loop -----------------------------------------------*/

INT frontend_loop()
{
   /* if frontend_call_loop is true, this routine gets called when
      the frontend is idle or once between every event */
   return SUCCESS;
}

/*------------------------------------------------------------------*/

/********************************************************************\

  Readout routines for different events

\********************************************************************/

/*-- Trigger event routines ----------------------------------------*/

INT poll_event(INT source, INT count, BOOL test)
/* Polling routine for events. Returns TRUE if event
   is available. If test equals TRUE, don't return. The test
   flag is used to time the polling */
{
    return 0;
}

/*-- Interrupt configuration --------------------------------------------*/

INT interrupt_configure(INT cmd, INT source, POINTER_T adr)
{
    switch (cmd) {
    case CMD_INTERRUPT_ENABLE:
        break;
    case CMD_INTERRUPT_DISABLE:
        break;
    case CMD_INTERRUPT_ATTACH:
        break;
    case CMD_INTERRUPT_DETACH:
        break;
    }
    return SUCCESS;
}

/*-- Event readout ------------------------------------------------------*/





INT read_galil_event(char *pevent, INT off){
  float *pdata, a;
  float *pspid;
  float *pacc;

  char buffer[500];
   char buffer1[500];
   char buffer2[500];
   hkeyclient=0;
   cm_get_experiment_database(&hDB, NULL);
   int size2 = sizeof(getaxes);
   INT size3 = sizeof(allow);

   //db_find_key(hDB,0,"/Equipment/Galil/Variables",&hkeyclient);
   db_get_value(hDB, hkeyclient ,"/Equipment/Galil/Variables/Setting",&ge
   db_get_value(hDB, hkeyclient ,"/Equipment/Galil/Variables/Condition",&
//read values from Condition and store it in variable allow

   //the variable allow is controlled by the user. Movement only starts
    sprintf(buffer ,"PAA=%d", getaxes [0]);
     sprintf(buffer1 , "PAB=%d", getaxes [1]);
     sprintf(buffer2 , "PAC=%d", getaxes [2]);
     if(allow==1){

  //send command to Galil

   GCmd(g, buffer);
   GCmd(g,"BGA");
   GCmd(g, buffer1);
   GCmd(g,"BGB");
   GCmd(g, buffer2);
   GCmd(g,"BGC");

   allow=0;

   db_set_value(hDB,0 ,"/Equipment/Galil/Variables/Condition",&allow , siz

     }







   rc = GMessage(g, buf1, sizeof(buf1));
   //cout << buf1 << endl;

   stringstream iss (buf1);
   // output returned by Galil is stored in the following variables
   iss >> axes[0];
   iss  >> axes[1];
   iss >> axes[2];
   iss >> speed[0];
   iss >> speed[1];
   iss >> speed[2];
   iss >> acceleration[0];
   iss >> acceleration[1];
   iss >> acceleration[2];
   iss >> torque[0];
   iss >> torque[1];
   iss >> torque[2];



   //upgrade ODB
  cm_get_experiment_database(&hDB, NULL);

  db_set_value(hDB, 0, "/Equipment/Galil/Variables/Position",&axes, siz
  db_set_value(hDB,0,"/Equipment/Galil/Variables/Speed",&speed,sizeof(s
  db_set_value(hDB,0,"/Equipment/Galil/Variables/Acceleration",&accelera
  db_set_value(hDB,0,"/Equipment/Galil/Variables/Torque",&torque,sizeof


    bk_init32(pevent);

    /* create banks */
    bk_create(pevent, "AXES", TID_FLOAT, (void **)&pdata);
    for (int j=0;j<3;j++){
      *pdata++ = axes[j];
    }

     bk_close(pevent,pdata);

     bk_create(pevent, "SPID", TID_FLOAT, (void **)&pspid);
     for (int j=0;j<3;j++){
       *pspid++ = speed[j];
     }
     bk_close(pevent,pspid);

     bk_create(pevent,"ACCL", TID_FLOAT, (void **)&pacc);
     for(int j=0;j<3;j++){
       *pacc++ = acceleration[j];
     }
     bk_close(pevent,pacc);
     return bk_size(pevent);
  }
```